UNIVERSITY OF CALIFORNIA
SANTA BARBARA

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

B2435635

CASE TECHNOLOGY AND THE SYSTEMS DEVELOPMENT LIFE CYCLE: A PROPOSED INTEGRATION OF CASE TOOLS WITH DoD STD-2167A

by

GARY THOMAS BATT

March 1989

Thesis Advisor: Barry A. Frew

Approved for public release; distribution unlimited.

T241660

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION CLASSIFIED	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
DECLASSIFICATION / DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION val Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 37
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
ADDRESS (City, State, and ZIP Code) nterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
NAME OF FUNDING / SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)
SE TECHNOLOGY AND THE SYSTEMS DEVELOPMENT LIFE CYCLE: A PROPOSED INTEGRATION OF SE TOOLS WITH DoD STD-2167A

PERSONAL AUTHOR(S)
tt, Gary Thomas

TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
ster's Thesis	FROM _____ TO _____	1989, March	74

SUPPLEMENTARY NOTATION
he views expressed in this thesis are those of the author and do not reflect the official osition of the Department of Defense or the U.S. Government.

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer Aided Software Engineering; Systems Development Life Cycle; DoD STD-2167A

ABSTRACT (Continue on reverse if necessary and identify by block number)

The use of Computer Aided Software Engineering (CASE) tools has been marketed as a medy for the software development crisis by automating analysis, design, and coding. The stems Development Life Cycle (SDLC) has been employed in an attempt to ease the velopment backlog by applying structured methods to the development of software systems. is study reviews CASE tool components and the future of CASE integrated toolkits, compares SDLC with the Defense System Software Development standard - DoD STD-2167A, and proposes means for integrating CASE tools into the DoD STD-2167A system development life cycle.

0 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
2a NAME OF RESPONSIBLE INDIVIDUAL rof. Barry A. Frew	22b TELEPHONE (Include Area Code) (408) 646-2924	22c. OFFICE SYMBOL Code 54Fw

D FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24.

Approved for public release; distribution unlimited.

CASE Technology and the Systems Development Life Cycle: A proposed integration of CASE tools with DoD STD-2167A

by

Gary Thomas Batt
Lieutenant, Supply Corps, United States Navy
B.B.A., University of Massachusetts, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1989

ABSTRACT

The use of Computer Aided Software Engineering (CASE) tools has been marketed as a remedy for the software development crisis by automating analysis, design, and coding. The Systems Development Life Cycle (SDLC) has been employed in an attempt to ease the development backlog by applying structured methods to the development of software systems. This study reviews CASE tool components and the future of CASE integrated toolkits, compares an SDLC with the Defense System Software Development standard - DoD STD-2167A, and proposes a means for integrating CASE tools into the DoD STD-2167A system development life cycle.

C.1

TABLE OF CONTENTS

I. INTRODUCTION ---------- 1
A. BACKGROUND ---------- 1
1. Software Development Crisis ---------- 1
2. Systems Development Life Cycle ---------- 3
3. Software Engineering ---------- 5
4. Systems Engineering ---------- 6
B. RESEARCH FOCUS ---------- 6
C. THESIS ORGANIZATION ---------- 8
II. CASE and I-CASE ---------- 9
A. CASE TOOLS ---------- 9
1. Definitions ---------- 9
2. Components of CASE Toolkits ---------- 11
3. Common CASE Features ---------- 11
B. I-CASE TOOLS ---------- 13
C. CASE LIFE CYCLE ---------- 14
D. THE FUTURE OF CASE ---------- 17
1. Reverse Engineering ---------- 17
2. Expert Systems ---------- 20
3. Quality and Productivity ---------- 21
E. CASE TOOL SELECTION ---------- 24
F. SUMMARY ---------- 25
III. DEFENSE SYSTEM SOFTWARE DEVELOPMENT/DoD STD-2167A -- 26
A. BACKGROUND ---------- 26

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5

B. DEFINITIONS ---------- 27
C. WHEN TO APPLY DoD STD-2167A ---------- 28
D. SOFTWARE DEVELOPMENT PROCESS ---------- 29
E. COMPARISON OF DoD STD-2167A TO THE CLASSIC SDLC - 32
1. Analysis and Design ---------- 32
2. Coding, Integration and Testing ---------- 34
3. Installation and Operations ---------- 36
F. SUMMARY ---------- 37
IV. DoD STD-2167A UTILIZING CASE ---------- 38
A. PROBLEMS WITH SDLC DEVELOPMENT ---------- 38
B. SYSTEM LIFE CYCLE ---------- 39
C. DEVELOPMENT USING CASE AND DoD STD-2167A -------- 42
1. DoD STD-2167A using CASE with IRDS ---------- 43
2. DoD STD-2167A using CASE without IRDS ------- 52
D. SUMMARY ---------- 55
V. CONCLUSIONS ---------- 57
A. DoD STD-2167A AND CASE ---------- 57
B. AUTOMATED TAILORING OF DoD STD-2167A ---------- 59
C. AREAS FOR FURTHER RESEARCH ---------- 60
1. Test of Proposed DoD STD-2167A ---------- 60
2. British Aerospace Australia ---------- 60
3. Productivity Measures ---------- 60
4. Expectations for CASE ---------- 61
LIST OF REFERENCES ---------- 63
INITIAL DISTRIBUTION LIST ---------- 65

LIST OF FIGURES

1-1. Classic Water-fall System Development Life ----- 4

2-1. The CASE Software Development Life Cycle ------- 15

2-2. Prototyping ---------- 16

2-3. Reverse Engineering Life Cycle ---------- 19

3-1. DoD STD-2167A System Development Life Cycle ----- 30

3-2. Classic Water-fall System Development Life ----- 31

4-1. Proposed DoD STD-2167A Life Cycle Using CASE ---- 44

ACKNOWLEDGEMENT

I wish to dedicate this thesis to my wife, Patricia. Without her assistance in improving my writing skills, her support in my pursuit of education, and the requirement placed on her to provide a single parent home during my last quarter, this document would not exist. I hope that I am able to provide her with equal support while she pursues her goals in the legal profession.

I. INTRODUCTION

Software development methodologies have been touted as a means of decreasing a large backlog in software development. Computer Aided Software Engineering (CASE) tools have been marketed by their developers as the means to shorten development times, thereby allowing more time to reduce the backlog. These two concepts are not mutually exclusive, but mutually dependent. This thesis will explain how CASE tools can be effectively integrated into the development cycle. The combined effect of CASE tools and sound development principles should enable accelerated software development and lead to easing what has been referred to as a "Software Development Crisis." An in-depth discussion of CASE tools is provided in Chapter II.

A. BACKGROUND

1. Software Development Crisis

Data processing systems are often viewed by users as a chaotic mess of redundant data that cannot provide required information in a timely manner [Ref. 1:p. 5]. Additionally, application developers commonly have backlogs of several years. Maintenance for operational systems is being estimated to take 50 to 80 percent of analyst and programmer time [Ref. 2].

The extent of the problem really is best expressed in three pertinent statements:

1. ...25 percent of the draft age population will be required to maintain DoD software by the year 2000. [Ref. 3]

2. ...the national demand for software is rising by at least 12 percent per year, while the supply of people who produce software is increasing about four percent per year and the productivity of those software producers is increasing at about four percent per year; this leaves a cumulative four percent gap. [Ref. 4:p. 31]

3. Not much progress has been made in the past 20 years in getting rid of three- to five-year backlogs of projects and for new applications and major enhancements to existing ones. [Ref. 5:p. 38]

Unfortunately researchers don't know whether the problem with software development is due to existing systems having been poorly developed and therefore requiring more maintenance time, or whether users are demanding applications faster than developers can produce them. What is known is that software engineering techniques can decrease the software backlog.

CASE has been declared by some to be the silver bullet that will save the software industry. Few agree that it is a cure-all for the problem, but in James Martin's words, "I-CASE[1] is THE most important change in professional computing practice in three decades...." [Ref. 3] How CASE and I-CASE technology will affect the development of

---

[1] I-CASE is a hybrid version of CASE, Integrated-CASE. I-CASE combines multiple CASE tools into a single architecture. Further discussion of this is presented in Chapter II.

computing systems, particularly by those systems constrained by DoD STD-2167A, is the focus of this thesis.

2. Systems Development Life Cycle

Traditional development of computer systems follows a waterfall sequence known as the Systems Development Life Cycle (SDLC), as presented in Figure 1-1. Depending on the author referred to, there are between five and eight major phases to the SDLC. This thesis will refer to a seven phase life cycle:

1. Investigation phase
   - Initial investigation
   - Feasibility study
2. Analysis and General Design phase
   - Existing system review
   - New system requirements
   - New system design
   - Implementation and installation planning
3. Detailed Design phase
   - Technical design
   - Test specification and planning
   - Programming and Testing
4. Implementation phase
   - User training
   - System testing
5. Installation phase
   - File conversion
   - System installation
6. Review phase
   - Development recap
   - Post-implementation review
7. Maintenance phase [Ref. 6:p. 467].

The Department of Defense (DoD) has created a standard SDLC that is embodied in DoD STD-2167A. It encompasses all phases of the life cycle in a unique format

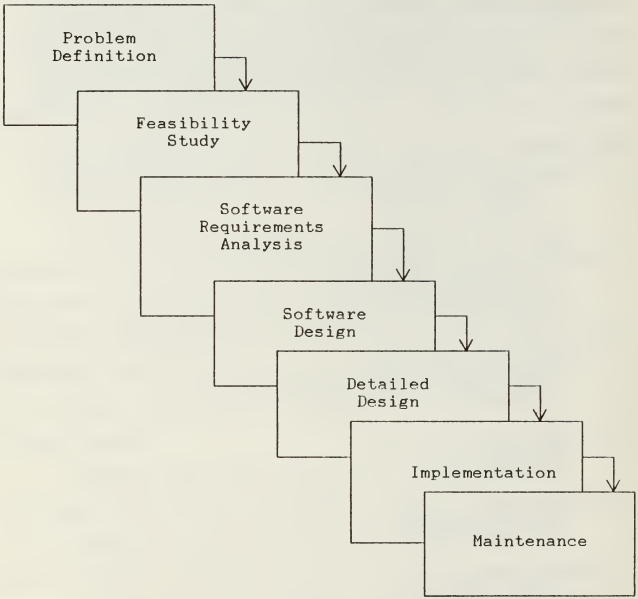

Figure 1-1. Classic Water-fall System Development Life

that requires an average of 200 words of documentation for each line of code produced [Ref. 7].

All current CASE tools fit into at least one phase of the SDLC, but most tools are designed to serve in the UPPER-CASE[2] arena. CASE tools that cover the entire life cycle are still a dream of designers, but by the mid to late 1990's, they should be a reality [Ref. 7].

3. Software Engineering

Software engineering has been defined as

> ...a set of three key elements - methods, tools and procedures - that enable the manager to control the process of software development and provide the practitioner with a foundation for building high-quality software in a productive manner. [Ref. 8:p. 19]

Although there are various graphics based techniques such as Data Flow Diagrams (DFD), entity-relationship models, and system structure charts that have enabled analysts and users to better communicate, exclusively using these tools does not constitute software engineering. Software engineering must embody all three elements - methods, tools, and procedures. To be successfully used in a software development organization, a sound methodology must be in place prior to introduction of CASE. [Ref. 9:pp. 17-20]

Software engineering principles allow developers and managers to create systems that fulfill user specifications. Its major drawback is that with large inventories of systems

---

[2] CASE tools designed to assist in the Investigation, Analysis and general design and Detailed design phases are being referred to as UPPER-CASE tools in trade publications.

to be maintained and new specifications to be programmed, software engineering doesn't answer the question, "Is the product specification correct for fulfilling our organizational goals at this time?" [Ref. 10]

4. Systems Engineering

Systems engineering encompasses all parts of software engineering plus methodologies for developing standardized system architectures. It embodies the trend toward end user computing and will allow information resource management ensuring that the specification fulfills the business requirement. [Ref. 10]

The trend in the use of CASE tools is to rectify the problems existing with software engineering by doing front end information strategic planning and business area analysis [Ref. 3]. CASE producers are beginning to recognize the need for automated code generation and expert knowledge of the corporate data structure by the CASE product [Ref. 11]. These changes propel CASE from Computer Aided Software Engineering to Computer Automated Systems Engineering.

B. RESEARCH FOCUS

CASE techniques offer a unique opportunity to decrease the backlog of applications development. They also present an opportunity to increase the quality of systems being developed through consistent use of a standard methodology throughout the life of the software.

The Department of the Navy has awarded an Umbrella Contract with Information Engineering Systems Corporation (IESC) for their CASE toolkit USER:Expert Systems™ and associated technical support. Many Navy and Marine Corps organizations are beginning to experiment with the toolkit to engineer their organizational structure and operations. [Ref. 12:p. 3]

The IESC toolkit addresses the concepts that James Martin refers to as information strategic planning and business area analysis. Information strategy planning is a top down look by senior management into the use of information by the enterprise. It builds an organizational model, identifies the information needs to support the strategic plans of the business, and establishes a framework for prioritizing the systems that are to be built for the organization.

Business area analysis is the next lower level of abstraction for the enterprise. Each business identified in information strategy planning is decomposed. Data and processes within the business are identified. A detailed model of each business that supports the overall enterprise is constructed.

Where the IESC toolkit leaves off after constructing the business model, this research continues. Specifically the thesis will answer the following questions:

1. Where within the detailed life cycle of DoD STD-2167A does CASE fit?

2. Do CASE tools need to be modified to fit within the construct of DoD STD-2167A or should the standard be changed to reflect the ever changing world of computer systems development?

C. THESIS ORGANIZATION

Chapter II presents an overview of CASE. A generalized model of the CASE life cycle is identified and compared to the prototype paradigm. A synopsis of the major characteristics of both CASE and I-CASE are offered along with summaries of what major contributors to the CASE industry perceive as the future of the tools.

In Chapter III, development and applicability to DoD STD-2167A are summarized. The life cycle standard for DoD STD-2167A is exhibited and compared to the classic waterfall life cycle of Figure 1-1.

Chapter IV introduces an effective scenario on how to use CASE for systems being developed under the guidance of DoD STD-2167A. A new model for the CASE systems development life cycle under DoD guidance is proposed.

Chapter V summarizes the contents of this work, addresses some of the drawbacks uncovered by implementing CASE, and proposes areas for further research regarding CASE and DoD STD-2167A.

II. CASE and I-CASE

This chapter provides an overview of the current Computer Aided Software Engineering (CASE) tools available, their principle operational features and components. Key definitions used within the CASE market and the CASE life cycle are presented. The chapter compares Integrated CASE (I-CASE) with current CASE technology and presents expert opinions on the future of CASE toolkit development. The chapter ends by offering a source for making CASE tool selection.

A. CASE TOOLS

CASE tools have been available in various forms to the professional software developer since the first compilers were introduced. CASE tools also include project management software, editors, prototype generators, version control coordinators, code re-engineering packages and other development programs too numerous to name here. Because of the wide range of tools being used to support systems development, the term CASE can be applied to any tool that helps users (of the tool) to develop programs.

1. Definitions

Some useful definitions of CASE terminology follow on the next page:

1. Application generator. These tools take design specifications and generate compilable code, usually in COBOL. Most application generator tools are used for building data base applications.

2. Data Flow Diagram (DFD). A graphic representation of the different data items in a system and their movement from process to process. DFDs depict the system from the data point of view. DFDs do not represent the control flow of a process.

3. Entity-relationship diagram (ERD). A diagram depicting objects and data elements, and the relationship between them. ERD are used to model the information and data in an organization.

4. Fourth Generation Language (4GL). A high level computer language which provides data base access facilities. 4GLs are easier to use than traditional languages such as COBOL and FORTRAN.

5. Leveling. The process of successively partitioning DFD parent processes into child processes in order to construct a hierarchically structured system.

6. Module. A collection of program functions that contains a set of routines with well-defined inputs and outputs.

7. Reverse Engineering. The methodology of taking an existing software system and decomposing the system into data elements and processes. The process is equivalent to taking apart a clock to find out what's inside and how it works.

8. Structure chart. A graphic tool that depicts the partitioning of a system into modules, showing the hierarchy and organization of those modules.

9. User interface. The end-user communicates with the application program through the user interface. User interfaces allow end-users to perform operations and view results.

10. Warnier-Orr diagrams. A data structure and file format showing the hierarchical structuring of substructures within a larger structure in an outline style. [Ref. 13:pp. 275-280]

2. Components of CASE Toolkits

All CASE toolkits contain a user shell that provides the user with a menu driven interface to the tools within the CASE software package. The following seven tools are frequently found in the various CASE toolkits, but no single toolkit on the market today has all seven:

1. Window, screen, report, graph and other output formatting editors.
2. Program flow editors including DFD's, traditional flow charts, and ERDs.
3. Schema design and data dictionary managers to build and maintain the CASE Data Dictionary.
4. Code management systems for version control and code maintenance.
5. Program development tools including 4GLs, prototyping tools, and application generators.
6. Bug reporting and tracking to allow automated program maintenance.
7. Network management tools. [Ref. 14:p. 40]

3. Common CASE Features

a. Data Dictionary

The single most important part of any CASE tool is its Data Dictionary. It unfortunately is also the part of the toolkit that frequently makes integration of toolkits impossible.

The Data Dictionary contains names and descriptions of the processes, data items, variables, access control lists for the dictionary and various other passive information [Ref. 1:p. 23]. The dictionary allows the system

designer to create, modify and delete various DFDs (or other graphical representations of the process) and data elements. It also should allow cross-referencing to objects defined in other systems designed using the same CASE tool.

CASE tools using Data Dictionaries are limited in their ability to integrate with other tools due to the Data Dictionary having no "understanding" of the system design. Data Encyclopedias have this capability and will be presented within the I-CASE subheading.

b. Visual/Graphic Representation

Whether the CASE tool uses DFDs, ERDs or System Structure Charts, it should combine graphical representation of the process with the textual representation in the Data Dictionary. Various levels of the abstraction for the process are retained. The visual display of the process is key to assisting users and analysts in specifying complete functional requirements.

c. Automated Consistency Checking

Throughout levels of the process, CASE tools ensure that naming conventions remain constant. Leveling is automatically performed and errors of omission are highlighted for rectification.

d. Multi-user Access

With most systems development processes, the project is incapable of being completed by one person. To assist in this, CASE tools provide tracking of those who have

access to a project and provides mechanisms to control access to all data elements and objects. Multi-user access will also allow designers of separate projects to re-use data definitions and functional modules that have been previously designed and stored in the Data Dictionary.

B. I-CASE TOOLS

I-CASE toolkits incorporate all of the best features of many CASE tools into a single package that is intended to cover the entire SDLC. Today's I-CASE tools do not approach the 100 or more functions that must be accomplished in the development of software systems, but many do perform 20 to 30 of them [Ref. 7].

The heart of an I-CASE toolkit is its Data Encyclopedia. Differing from a Data Dictionary, the Data Encyclopedia stores the meaning as well as the content of the entries. The encyclopedia accumulates knowledge about how and why a process is performed. Rules regarding how processes are to be linked, structures are to generated from DFDs, and data elements are to be referred to are stored during the development of the Data Encyclopedia. Rule processing is then used to achieve accuracy, integrity, validity and completeness of plans. The encyclopedia should grow over time to encompass the entire body of knowledge about the business processes of the organization. [Ref. 1:pp. 23-24]

An integral part of the I-CASE toolkit is automatic code generation from the knowledge maintained in the encyclopedia.

Its primary purpose is to create structured program modules fulfilling the functional requirements maintained in the encyclopedia. The code generation module probably would not be able to create optimal programs for an intended target machine, but code optimizers exist that are designed for specific hardware and software configurations.

C. CASE LIFE CYCLE

CASE proponents expound upon the need for an evolutionary life cycle, similar to the paradigm used for prototyping. The major difference between a CASE evolutionary life cycle, Figure 2-1, and a prototyping life cycle, Figure 2-2, is that CASE development will not create software until the design is engineered to meet user specifications.

CASE life cycle relies on the continued expansion of the Data Dictionary. Each software development phase will allow the designer to draw information from the Data Dictionary thereby reducing inconsistency. As the Data Dictionary increases its knowledge base, each software development project will be able to extract previously defined information thereby reducing development time and creating consistency across all software products.

Prototyping initially creates software for the user and continues to modify it until it meets the user specifications. Since most prototyping is done with less efficient programming languages and models only specific portions of the requirement, the software application is

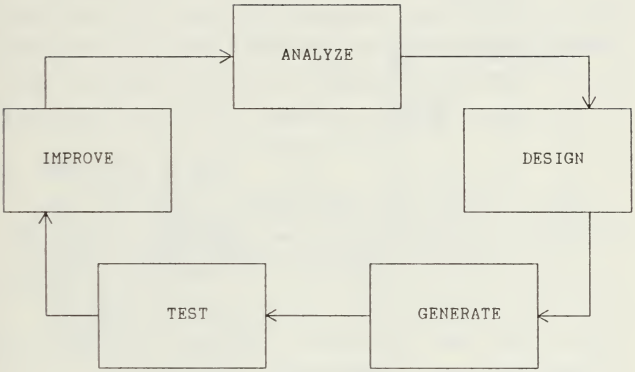

Figure 2-1. The CASE Software Development Life Cycle [Ref. 1]

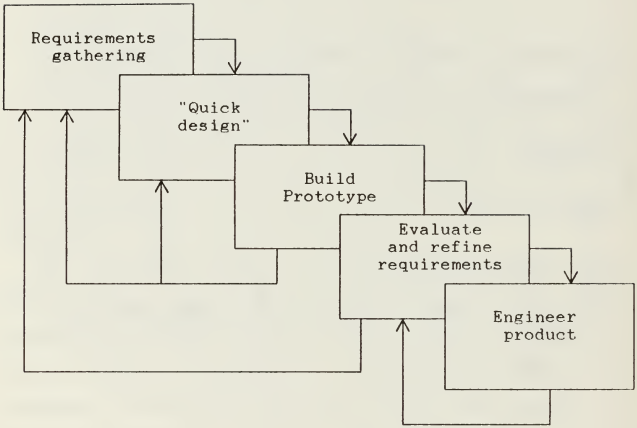

Figure 2-2. Prototyping [Ref. 8:p. 23]

usually re-coded and installed using third-generation languages. The prototype software product is considered a throw-away and the effort used to develop it becomes a formal requirements definition used to code, test, and implement as with any SDLC project.

Components of the prototype life cycle fit within the CASE life cycle. Specific prototyping tools, such as report generators and screen formatters, are considered to be CASE tools. The use of prototype tools in the CASE life cycle is discussed as part of the Design activity of Chapter IV.

D. THE FUTURE OF CASE

> ...the technological base on which one builds is always advancing. As soon as one freezes a design, it becomes obsolete.... The challenge and the mission are to find real solutions to real problems on actual schedules with available resources. [Ref. 15:p. 9]

Some CASE tools are in their infancy and other tools are more mature, but none today reaches what James Martin would call an I-CASE toolkit. The solutions to problems stated in the opening chapter will not come by passively waiting for I-CASE, but by following Fred Brooks' advice and solving today's problems with the toolkits that are currently available. What specific CASE tools will be available in the future is unknown, but experts' opinions of what is expected will be presented in the following pages.

1. Reverse Engineering

A significant investment in software systems already exists and is draining almost 80 percent of analyst and

programmer time to maintain it. Estimates of 77 billion lines of COBOL code in IBM production systems alone lead one to believe that the job of data manipulation is already being performed and that new applications development is not really required. So why is there a fuss over CASE to engineer new systems from scratch?

The CASE life cycle depicted in Figure 2-3 shows how reverse engineering fits. By starting at the operational level with an existing application, analysts identify end users, terminals, computers, record instances, and programs in execution that are required to enable the software to operate. This information is passed along to Database Administrators (DBA) and programmers at the implementation level to catalog the source-level descriptions of files, databases, and programs. Design objects identified include: records, sets, reports, screens, programs, and statements. The repository of information regarding the application is then passed to Data Analysts and Systems Analysts at the specifications level to extract the underlying data model required to operate the business. Objects identified include entities, relationships, processes, and procedures. At the highest level, the requirements level, Business Analysts identify the critical success factors, goals, requirements and organizations that the application has been satisfying. [Ref. 16:pp. 49-56]

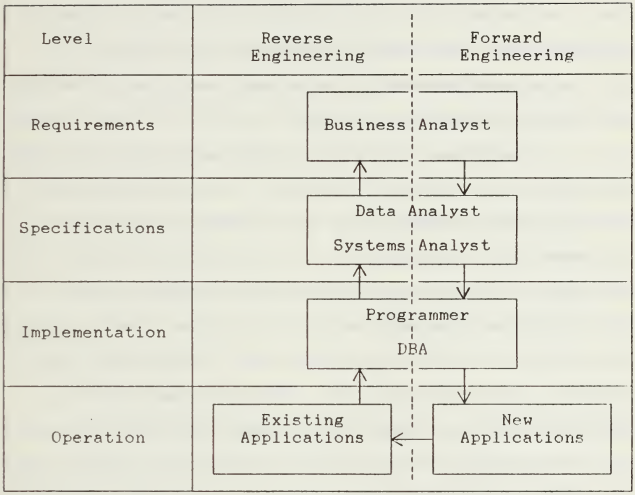

Figure 2-3. Reverse Engineering Life Cycle [Ref. 16:p. 50]

After all design objects have been identified, the application is re-engineered downward through the cycle taking advantage of a single corporate data encyclopedia. All new applications become existing applications as soon as they are put into operation. Enhancements to systems are always validated against the business requirements. Re-engineering existing software reflects the continuity of applications and allows for revisions to constantly reflect the current business architecture.

By targeting maintenance (both error correction and enhancement) of operational systems, next generation CASE tools open the possibility of reverse engineering the processes that currently exist. Although it would be preferable to have reverse engineering automated, human intervention will be required to fill in the holes. Unfortunately, existing applications do not document the goals for which the program was designed, where all of the manipulated data is targeted to be deposited (and why), and who is responsible for the maintenance of the accurate inflow and outflow of information.

2. Expert Systems

Within one year, expert systems built into the systems development and maintenance environment of available CASE tools should be able to identify all of the significant data in a business unit. This will include what each data element and structure means, what it depends on and what

depends on it, where the physical data is currently located, and who is authorized to use, modify, create, and delete that data. The tool will be programmed to understand the functions and text of all current modules of code. It will continuously optimize its own performance without human intervention. [Ref. 11]

An expert system within the CASE tool will support the transition to new application development becoming nothing more than a special form of maintenance. Each time a developer invokes the CASE tool to start a new project, the expert system will associate the data elements and processes to existing systems within its encyclopedia. Employing an interactive dialogue between the developer and the expert system, development becomes nothing more than modification of existing modules of code and manipulation of previously defined data elements. [Ref. 11]

The ability of a CASE toolkit to do this will hinge on a strong management commitment to develop a corporate data encyclopedia. The CASE toolkit must also be integrated as previously described in the paragraphs on I-CASE.

3. Quality and Productivity

Software quality and productivity in systems development are two of the hardest entities to define. What is a quality software product? Is it an application that has zero bugs when it is implemented or an application that meets all of the users' specifications? The latter definition

intuitively is more acceptable, but it is not necessarily correct. That definition loses its usefulness if the user specifications do not fulfill the business' strategic goals. Therefore, the application may give the results requested by the user, but it has tied up development time and money on a project that will not serve to enable the enterprise to achieve strategic advantage.

To paraphrase Vaughan Merlyn, CASE is about breaking down the communications barriers between users and designers [Ref. 10]. Through the use of graphics interfaces, CASE tools allow users and designers to "see" the structure of an application from the top down. By approaching systems development from the total organization perspective and using integrated tools driven by a data encyclopedia, all applications are developed within a framework of how they fit into the overall business strategy.

So where do quality and productivity fit into the future of CASE? The traditional view of the quality/productivity relationship has been an inverse relationship. It has been assumed that as quality increased, the costs of achieving that quality increased - fewer errors in a program could only be achieved by increased time to debug the software and an increased probability that the application would not be completed in time. It was also assumed that as programmers sought to increase the number of lines of code produced, an increase in the number of errors

would be introduced and the quality of the documentation decreased. The reality of the relationship is that as quality increases, productivity usually follows. An example of this comes from Japan.

> Transistor radios were laughed at by the world's markets in the 1960's. Today, thanks to a concerted effort by Japanese management, the quality of all electronics far outstrips the remainder of the world's suppliers. Japan's electronics industry simultaneously experiences the highest productivity in the world. [Ref. 10]

By introducing an engineering approach to systems development, as is required by the utilization of CASE, quality becomes designed into the application. An engineered product relies on correctness, consistency, completeness, and coherence. These factors will reveal all levels of the structure and improve the quality of the system architecture. The approach will also reduce the introduction of defects during design, improving the quality of execution of the application.

Productivity gains associated with the introduction of CASE should be measured against current systems development statistics. Currently 25 percent of major systems are cancelled before implementation, three to ten years are required to develop major systems, and maintenance costs are approaching 80 percent of major systems' overall budgets. In a full CASE environment, which is not expected until 1995, less than 10 percent of major systems will be cancelled, one to two years will be required to develop major systems and maintenance costs will be reduced by 75 to 90

percent below the current 80 percent figure. [Ref. 7, Ref. 17]

Other significant measures are important for raising the awareness of how CASE can positively affect systems development productivity in the future. In developing a 1000 Line of Code (LOC) product, today's development schedule would call for one month. That same product could be produced in one day under I-CASE, a 20:1 improvement. Cost savings for the application show a similar 20:1 improvement with today's development cost at $5000 and I-CASE development cost of $250. However, the improvement ratios are reduced below 5:1 in development time and costs for a 10,000,000 LOC product due to multi-member development teams and the inherent communications costs. [Ref. 7]

E. CASE TOOL SELECTION

The proper methodology for selecting a CASE tool today has been documented in several articles, one of the best being "A Guide to Selecting CASE Tools" by Michael Gibson which appeared in the July 1, 1988 issue of DATAMATION. The fact that over 100 vendors offer CASE products makes the selection process extremely difficult. What is optimal for one organization is not always optimal for another. The toolkit selected must be dynamic enough to support the complete spectrum of the SDLC and also be able to be integrated with other CASE tools, Data Base Management Systems, and hardware configurations. A design tool that

leaves the developer with DFDs and a Data Dictionary that must be manually converted to COBOL picture statements and executable code is barely better than doing the whole job manually [Ref. 3].

F. SUMMARY

In the relatively short life of CASE tools, they have displayed promise for alleviating the software crisis. From their humble beginnings as a graphical representation of the familiar Yourdon/DeMarco structured analysis and design methodology or Warnier-Orr diagrams, CASE tools have developed to the point of being interactive complete life cycle support tools. With the rapid integration of expert systems into CASE tools, the probability that they will truly automate systems development exists. Computer Automated Systems Engineering is on the visible horizon.

III. DEFENSE SYSTEM SOFTWARE DEVELOPMENT/DoD STD-2167A[3]

This chapter provides an overview of DoD STD-2167A, Defense System Software Development, summarizes key definitions used by the standard, and describes its applicability to DoD software projects. The chapter compares the individual phases of DoD STD-2167A with the phases of SDLC, as introduced in Chapter one.

A. BACKGROUND

DoD STD-2167A is the first major revision to DoD STD-2167 and superceded it as of 1 April 1987. The requirement was developed in conjunction with DoD STD-2168, Software Quality Program, to ensure that standards were maintained in the development of software systems. Both standards establish a well defined and easily understood software development and acquisition process. The standards were intended to supercede all existing DoD standards, reducing confusion and eliminating conflicts.

DoD STD-2167A, a simplified version of its predecessor, allows more latitude in the development process. The revised standard reduced the number of Data Item Descriptions (DIDs) from 24 to 19. (The Definitions section of this chapter provides more information regarding DIDs).

---

[3] The contents of this chapter, unless otherwise indicated, were drawn from [Ref. 17].

DoD STD-2167A allows tailoring by eliminating non-applicable requirements and permits the developer to practice their own software development methodology. The standard is compatible with modern methods of software development, and it supports rapid prototyping if the Software Development Plan (SDP) is tailored and specifies that methodology. DoD STD-2167A is intended to focus visibility on the software development and acquisition process throughout the life cycle by formal requirements reviews and audits at the completion of all milestones. The standard should be applied throughout the life cycle and should provide cost benefits throughout.

B. DEFINITIONS

The following definitions will enable the reader to better understand the contents of this chapter:

1. Computer Software Component (CSC). A distinct part of the software product. It can be equated with one program of a complete software system ie., the check printing program of a payroll system.

2. Computer Software Configuration Item (CSCI). The complete software system.

3. Computer Software Unit (CSU). A single module of the CSC that can be tested for functional accuracy.

4. Data Item Descriptions (DIDs). DIDs describe the set of documents for recording information required by the standard.

5. Developmental Configuration. The software and associated documentation that defines the evolving configuration during development. The Developmental Configuration consists of a Software Design Document and source code listings.

6. Product Baseline. The software as designed, tested and implemented prior to installation.

7. Software Development Plan (SDP). A single document outlining the steps for conducting the activities required by the standard.

For a complete listing of the definitions applicable to DoD STD-2167A the reader is referred to Chapter three of the standard.

C. WHEN TO APPLY DoD STD-2167A

This military standard is approved for use by all Departments and Agencies of the Department of Defense. Its intended use is for the acquisition, development or support of software systems. Either a commercial enterprise under contract to the government or a government agency that performs software development can be substituted for the term "contractor" where the standard specifies. This standard is planned to be used in conjunction with MIL-STD-499, Engineering Management, for total system development.

DoD STD-2167A is required for use in all mission-critical systems development projects. Mission-critical projects include:

1. Intelligence activities
2. Command and control of military forces
3. Cryptologic systems relating to national security
4. Equipment or software forming an integral part of a weapons system.

Use of DoD STD-2167A is not required, although it is encouraged, on other systems development projects unless it

is specified in the contract. DoD STD-2167A does not apply to the development of hardware systems.

D. SOFTWARE DEVELOPMENT PROCESS

The contractor will implement a process for managing the development of deliverable software. The process will include the following activities, which may be overlapped or applied iteratively:

1. Systems Requirements Analysis/Design
2. Software Requirements Analysis
3. Preliminary Design
4. Detailed Design
5. Coding and CSU Testing
6. CSC Integration and Testing
7. CSCI Testing
8. System Integration and Testing
9. Testing and Evaluation
10. Production and Deployment

Figure 3-1 illustrates the standard software development waterfall as specified by DoD STD-2167A. (For clarity, Figure 1-1 has been repeated as Figure 3-2 on the following page and the hardware development processes edited from the DoD STD-2167A life cycle). The figure specifies the points where formal reviews and audits are to be completed. All reviews and audits specified are detailed in DoD STD-1521, Technical Reviews and Audits for Systems, Equipments, and Computer Software. Definitions for baselines can be

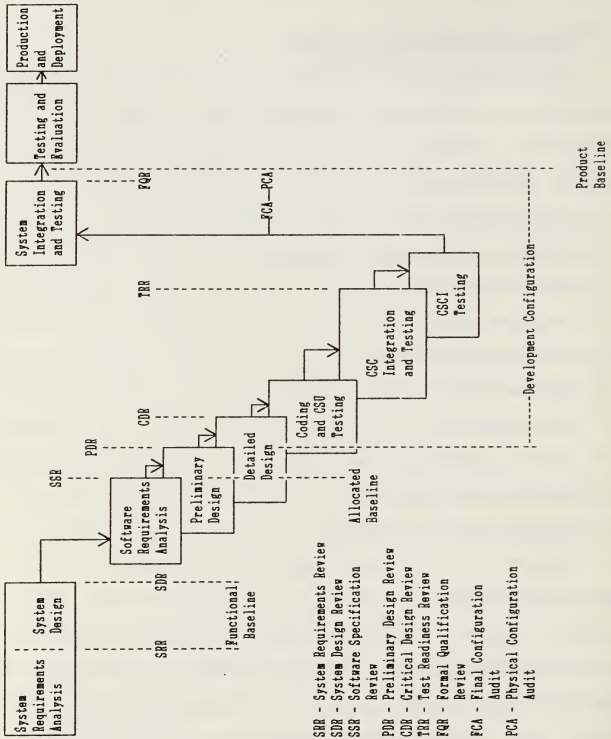

Figure 3-1. DoD STD-2167A System Development Life Cycle

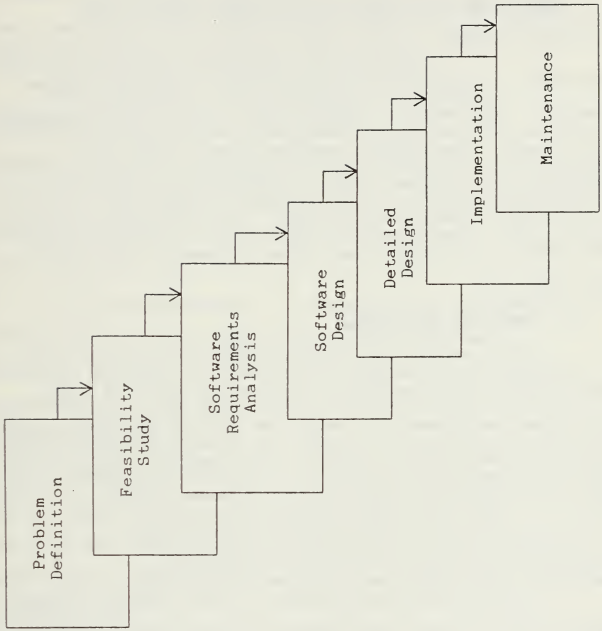

Figure 3-2. Classic Water-fall System Development Life

found in DoD STD-480, Configuration Control - Engineering Changes, Deviations, and Waivers.

E. COMPARISON OF DoD STD-2167A TO THE CLASSIC SDLC

DoD STD-2167A relies on all of the principles that have made SDLC a successful methodology for developing software systems. Both methods require that a step-by-step procedure be followed. The methods require formal reviews at completion of various steps and integrated testing prior to deployment of operational systems. DoD STD-2167A requires the completion of many more formal documents than does the SDLC approach.

1. Analysis and Design

Prior to any actual development of systems, heavy emphasis is placed up-front on gathering complete user functional requirements. While the SDLC Investigation phase involves conducting an initial investigation of the problem area and feasibility study of the proposed solution, DoD STD-2167A requires that in the System Requirements Analysis/Design phase five DIDs be completed.

The Systems Specifications, System Design Document, Preliminary Software Requirements Specifications, Preliminary Interface Requirements Specification, and Software Development Plans DIDs are to be completed prior to the System Design Review. Within the DIDs cost/benefit analysis, quality control, testing criteria, delivery timetables, software requirements, interface requirements and software

engineering methodologies are documented. Upon acceptance of these DIDs at a System Design review, a Functional Baseline is established. From this point on, any changes to the specifications or development plans must be approved through formal configuration management procedures.

The resultant single document of the SDLC Investigative phase is a cost/benefit analysis that will be reviewed formally by management. This phase usually involves one week to one month of analysis and is a check-point for determining whether to continue with the project. [Ref. 19:p. 47]

In the second phase of DoD STD-2167A, Preliminary Software and Interface Requirements are finalized. Additional analysis in these areas completes the software specifications. The specifications are reviewed formally against the system specifications and in accordance with the Software Development Plan at a Software Specification Review. When accepted, software development commences with specifications that are signed-off as having met the user requirements. This specification is considered the Allocated Baseline.

After a commitment to continue an SDLC project, the second phase, Analysis and general design, compares closely to DoD STD-2167A in that the new system specifications are finalized. This only occurs after a complete review of the existing system. DoD STD-2167A does not require review of

existing systems, although most analysts will do so informally. SDLC also completes implementation and installation planning during this phase, steps that were required as part of the SDP by DoD STD-2167A.

In the second phase of SDLC, an overall new system design is documented. Technical designs are postponed until the third phase. DoD STD-2167A phase three, Preliminary Design, is a single stage that accomplishes the equivalent of the general software design. Preliminary Software Design, Software Test Plan, and Preliminary Interface Design DIDs are completed. At the completion of this phase, the Preliminary Design Review is accomplished and the Development Configuration is established.

Detailed design occurs in the fourth phase of DoD STD-2167A and the third phase of SDLC. Technical software design is completed, interface details are documented, and testing plans are formalized at this point. DoD STD-2167A completes this phase with a Configuration Design Review which formalizes and approves the detailed design as documented in the Software Design Document, Software Test Description, and Interface Design Document DIDs. SDLC includes programming and testing in this phase, which will be covered in the next section of this chapter.

2. Coding, Integration and Testing

SDLC integrates coding and testing into the third phase of the software life cycle. Although the SDLC does not

formalize how coding and testing should be done, it usually follows closely to the methods utilized by DoD STD-2167A. The DoD standard is divided into three phases in this area, Coding and CSU Testing, CSC Integration and Testing, and CSCI Testing.

Within Coding and CSU Testing, the development team usually takes individual program modules and begins programming them from the lowest level modules -- a bottom-up approach. When programs of individual modules are completed, they are tested for programming errors and corrected.

When the modules of a CSC are completed, they are integrated and tested in the CSC Integration and Testing Phase. At the completion of this phase, a Test Readiness Review is conducted to review Source Code Listing(s), Source Code, and Software Test Description DIDs. The Test Readiness Review is intended to confirm that the modules are adequately prepared and documented in accordance with the system specifications prior to conducting CSCI Testing. Completion of CSCI testing requires that Updated Source Code, Software Test Report, Operation and Support Document(s), Version Description Documents, and Software product Specification DIDs be completed prior to beginning the next phase.

The fourth phase of SDLC, Implementation, is really not an implementation phase. During this phase the users are trained to use the software system and the system is tested. DoD STD-2167A phase System Integration and Testing performs

only the testing functions. At this stage both life cycle methodologies require that testing be accomplished on hardware that is equal to the production hardware. Equality of the testing hardware is determined by the contracting agency for DoD STD-2167A contracts and by the user for SDLC projects. Additionally, DoD STD-2167A software must be independently tested by an activity not associated with the developer to ensure that the software meets the specifications. In most cases, Functional and Physical Configuration Audits are conducted after this phase, although they are scheduled to be completed after CSCI testing. A Functional Design Review is conducted to establish the Product Baseline.

3. Installation and Operations

The Installation phase of SDLC requires that all existing system files be converted to the new software and the software be installed in the production environment. The Review phase involves a complete evaluation of the development and a post-implementation review. DoD STD-2167A conducts these steps in the Testing and Evaluation phase, but may or may not have installed the software at sites that will be operating the software system. Evaluation of the software development occurs prior to deployment of the product.

The final phase of DoD STD-2167A is Production and Deployment. In this phase the software is installed in the production environment and users are trained to use the

system. SDLC's final phase is maintenance of the software product. DoD STD-2167A does not consider maintenance as part of the Life Cycle and that problem is addressed in Chapter IV.

F. SUMMARY

DoD STD-2167A relies on a proven structured development life cycle technique. It compares readily to the classic waterfall life cycle which has been used extensively in developing both large and small software products. The waterfall life cycle has inherent problems that will be discussed and addressed in the next chapter. The next chapter also proposes a combination of CASE toolkit methods, DoD STD-2167A requirements, and Information Engineering approaches that should aid in easing the software development crisis.

IV. DoD STD-2167A UTILIZING CASE

DoD STD-2167A allows developers to utilize whatever software methodology that is productive for them as long as it is documented in the Software Development Plan. The opening sections of this chapter provide a discussion of the problems inherent in SDLC methodologies and differences between a development cycle and system life. A proposal of how CASE tools can be integrated into the development cycle for software being developed under the requirements of DoD STD-2167A follows.

A. PROBLEMS WITH SDLC DEVELOPMENT

The first problem with SDLC methodologies is that they do not promote interaction between the developers and users after the analysis phase. Both SDLC and DoD STD-2167A require that formal reviews be conducted at the completion of the phases, but these reviews are usually conducted by management, not the users. [Ref. 14:pp. 4-7]

The second problem with waterfall development life cycles is they tend to use a bottom-up approach to implementation. The system is not completed until all modules are operational. DoD STD-2167A exemplifies this in the following development approach: (1) CSU coding takes place, (2) CSUs are tested, (3) CSC testing takes place, (4) CSCI integration and testing take place. Each module is sequentially added to

an operational module attempting to test module interfaces and requiring debugging at each module interface. By employing a top-down approach of first coding and testing interface modules with dummy modules performing the operations of CSUs, the need for interface debugging would be significantly reduced. Furthermore, the systems could be incrementally installed with the modules that are operational. [Ref. 19:pp. 47-56]

The third problem with waterfall development life cycles is that there is a tendency to follow a sequential path [Ref. 19:p. 48]. Although DoD STD-2167A states that the contractor may follow whatever methodology it wishes, if it is properly documented, the completion of DIDs and their acceptance at formal reviews precludes the developer from proceeding to the next phase.

B. SYSTEM LIFE CYCLE

Proponents of life cycle development methodologies place major emphasis on the process to be followed in developing systems and software. The requirement to follow a procedure, a development life cycle, to ensure continuity between steps in a development project is the goal of SDLC. However, software systems should be considered under the criteria of both "how" the system is developed and "how long" the system (system life) is supposed to fulfill its requirements. An analogy will be used to explain the difference between development cycle and system life.

Automobile manufacturers develop modes of transportation. The engineers in charge of a project take a general concept, the drivers requirement to get from place to place, and using previously designed and newly designed parts, assemble them into an automobile. The engineers second objective is to meet the peripheral requirements of users: seating capacity, leg room, acceleration, braking distance, etc. The engineers will continue to use previously designed parts to create the automobile and develop new parts only if required. The automobile is prototyped and tested thoroughly prior to mass assembly. This is a shortened version of the development cycle of the automobile.

The user decides on which automobile to purchase based on his requirements at the time. As the user's needs change and parts wear out, the automobile is modified and corrected. If the tires go bald, they are replaced with new tires. If the user purchases a boat, a trailer-hitch is installed. When the automobile can no longer provide adequate seating capacity for the owner (two seat sports car with a new baby in the family) or its frame rusts out, the automobile is replaced. Although the automobile is no longer serviceable to the user, miscellaneous parts of the automobile may be. The old tires, headlights, radio and other salvageable parts can be sold for use by another automobile. This is a shortened version of the life of an automobile.

This analogy applies readily to software development. When developing a software product, the engineer should be able to use previously designed parts, software modules, and data elements to fulfill the basic requirements of the user. The software engineer continues to add parts until the peripheral requirements are met and should only design and build new parts if they don't already exist. After thoroughly testing the assembled modules, the prototype, the software should be placed into production.

The term prototype should be redefined more in line with other engineering disciplines. A more suitable software engineering definition might be: a completely working model that meets user requirements and undergoes testing prior to being placed in a production situation. As the term is currently used, a prototype is more of a skeleton meeting basic user requirements that will be refined, added to, and built upon by more thorough analysis and design. Current prototyping is design and modification through trial and error, not development and testing.

The users' requirements will most likely change during the product's life. These changes could be faster processing requirements, new information needs, or reformatting. If the changes are replacements for wornout modules, they should be replaced with new modules. If the changes go beyond the original specifications of the software or the changes can't make the software perform user requirements, the software has

fulfilled its useful life and a new software system should be designed.

Software life should be similar to automobile life. Just as an owner of an automobile would check the specifications prior to adding a trailer-hitch in order to pull a ten ton yacht, the user of a software product (and the designer who will create the modification) should check the specifications of the software prior to adding upgrades. If a problem can be corrected with new parts and the basic system requirements are still being met (the "chassis" is sound), it will be corrected. If the software no longer meets the user's basic requirements, the usable parts will be salvaged for use by other software products and a new software product developed to replace it.

C. DEVELOPMENT USING CASE AND DoD STD-2167A

The requirements of DoD STD-2167A are an excellent place to begin creating a standard methodology for software developers to follow. The rigors imposed by the standard force developers to completely trace system requirements through development and testing. By utilizing CASE tools, as described in Chapter II, in the development process, many functions that are required by the standard could be automatically handled. Two scenarios of how CASE could be utilized under DoD STD-2167A are described on the following pages.

1. DoD STD-2167A using CASE with IRDS

Sharon Stanley describes in [Ref. 12] how Naval Supply Systems Command (NAVSUP) and Naval Sea Systems Command (NAVSEA) have utilized CASE tools to develop corporate Data Dictionaries using Information Engineering methodologies. The NAVSUP process of developing the Information Resource Dictionary System (IRDS)[4], as the corporate Data Dictionary is now known, continued to evolve with strategic policy documentation completed and business area analysis still in process until the departure of the Data Administrator at the end of January 1989. No formal plans to utilize the IRDS in software applications development now exists.

Figure 4-1 proposes how CASE tools with Data Encyclopedias fit into the structured development cycle of DoD STD-2167A. It also completes the cycle of how software life should be accounted for by DoD STD-2167A.

Given an environment with a partially or completely developed IRDS, software application development should be easier. Since CASE tools rely heavily on having a Data Encyclopedia, the IRDS becomes an integral part of the development environment. To avoid redundant storage of file, program, record, module, and element descriptions, many of the definitions in the IRDS will be used in the project Data Encyclopedia. The system designer will have to ensure that

---

[4] Ref. 20 provides a detailed description of IRDS and NAVSUP's development of their IRDS.

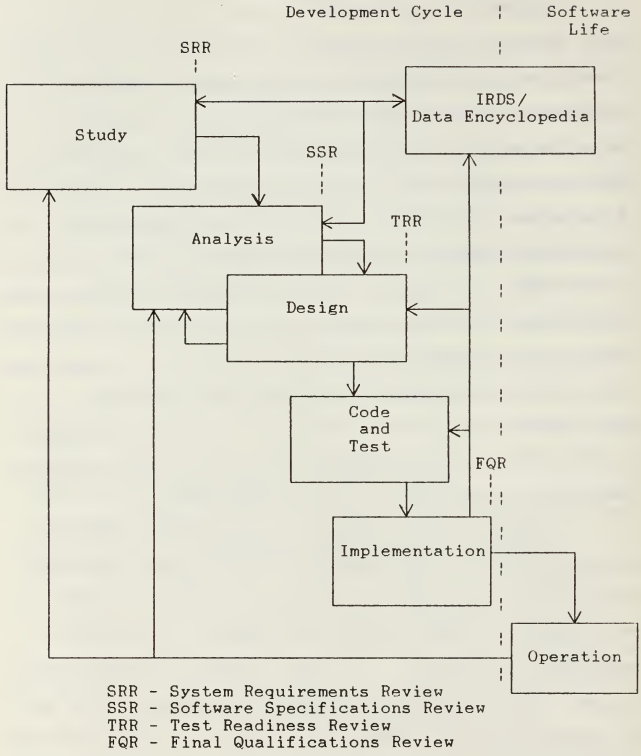

Figure 4-1. Proposed DoD STD-2167A Life Cycle Using CASE

selection of the CASE tools to be used will have import and export facilities to the IRDS. It must be emphasized that a major benefit of using CASE tools is the production of documentation by the tool. To this extent, when selecting CASE tools for use with DoD STD-2167A, several CASE products, including NASTEC's CASE 2000/DESIGNAID and Index Technology's EXCELERATOR, produce documentation for DIDs in DoD STD-2167A format.

a. Study Activity

The first activity in development of a system, the Study activity, is to get an understanding of what the users need. Using a CASE development tool that documents processes in the Data Encyclopedia in a Structured English format, the analyst and user should assess the system requirements, the policy as to what the system is to accomplish, and the deficiencies inherent in the current way of doing business. Using these elements, the analyst should compare similar processes and user requirements in the IRDS to find out if a system exists that can fulfill the goals.

The Study activity is a direct replacement for the System Requirements Analysis/Design Phase of DoD STD-2167A and will produce a System Development Plan. The SDP will outline the methodology to be used and contain an initial cost-benefit report. The SDP should be reviewed at a System Requirements Review between developers, users and management.

b. Analysis Activity

The Analysis activity in creating a software system should rely on a detailed investigation of what users need to fulfill their business requirements. Using a CASE diagramming tool, the Analysis activity will rely on as its starting point the organizations goals and constraints as well as the SDP established in the Study activity as the starting point. The analyst and users will develop a complete logical model detailing an environmental and a behavioral model. The environmental model details how the system interacts with the other systems and with users. The behavioral model specifies in complete detail what the system does. [Ref. 19:pp. 82-87]

Behavioral model development should utilize processes, data definitions, and relationships that were previously defined in the IRDS or the development should create processes, data definitions, and relationships that were not previously documented. In either case, during creation of the behavioral model, the model will be validated against business policy.

Using the logical model and the SDP, the analyst should develop new physical models for the user to review. The various new physical models will be evaluated at a System Specification Review. The users and management will choose between the options presented. Software Requirements

Specification and Interface Requirements Specification for the chosen model will be accepted as the Allocated Baseline.

During this part of Analysis, the Design activity will be overlapped because user, management, and operational constraints will be introduced that will limit design choices. The benefit of using a CASE tool will be that the design constraints originated during Analysis will be carried forward consistently into the next activity.

c. Design Activity

Using the Software Requirements Specification, Interface Requirements Specifications, and the process model stored in the CASE Data Encyclopedia, the design team should decompose the process model into a structure chart of the modules. The modules should be compared to modules stored in the IRDS to take advantage of previous development efforts.

With prototype tools that format screens and generate reports, the designers and users will design user-system interfaces. The data elements that are displayed within the prototype forms will be derived from data elements stored in the IRDS. If not there, the IRDS must be updated or the data element verified for accurate usage. Prototyping tools that generate code should be avoided because the code is specific to the application, is not modularized, and frequently cannot be used in other applications without significant modification.

Using CASE test generation tools, a set of test scenarios should be designed. The test cases should stress the system to ensure that it meets all performance criteria. The cases should test all individual modules and all man-machine or system-to-system interfaces. The test cases are integrated into a coordinated testing plan that is stored in the Data Encyclopedia as the Software Test Description for use during the Code and Test and the Implementation activities. The test cases should also be reviewed for accuracy and completeness at a Test Readiness Review. Acceptance of the testing program for the software design signals the end of the iterative Analysis and Design activities. At this point all software and interface requirements must be met or deferred to new development effort.

The Design activity may require further input by users, management, and operations. It is important that all data gleaned during design that is rightly attributed to the Analysis activity be documented in the Data Encyclopedia, exported to the IRDS, and updated in the System Requirement and Interface Requirement Specifications.

d. Code and Test Activity

The developers initial responsibility in the Code and Test activity is to determine the order in which to code modules. Using a top-down approach, coding should start with interface modules, as discussed in Chapter three. The

highest level modules will be coded and tested as CSUs. They are then linked and go through integrated testing with lower level modules as call and return dummy CSUs attached to the upper level modules. Individual lower level CSUs will be coded and tested for each internal action on the input data ensuring the accuracy of the output data (white box testing). When a CSU has completed testing, it is added to the upper level structure as a black box replacing the dummy CSU. The coding should be done with CASE application generators, and all testing should utilize the previously developed test cases. Completed modules should be exported to the IRDS.

When a skeleton system is complete, integrated testing should be started. An informal user review at this junction will enable designers to ensure that the interface requirements have been met.

Modules are to be coded, tested, and added to the skeleton system as completed. When the complete software system is coded and has been tested thoroughly by the developers, it should be tested on the actual destination hardware in stead of similar hardware as DoD STD-2167A requires. If possible, further testing of what is called the prototype system should be accomplished by an independent organization.

The DoD STD-2167A DIDs completed by this activity will be the Source Code, Source Code Listing, Operation and Support Documents, Version Description Documents, and

Software Product Specification. User training should start when the completed system is given to the independent testing organization.

e. Implementation Activity

The Implementation activity requires data conversion from the old to the new system. Also, this activity requires more user training. During this activity, the user and developer should review the development documents against the Software Development Plan and create a Lessons Learned Document for storage in the Data Encyclopedia. All elements in the Data Encyclopedia should be reviewed and exported to the IRDS.

The final step in this activity will be to conduct a Final Qualification Review. This review will act as the formal acceptance of the prototype software system. The acceptance of the software ends the development cycle. When placed into operation, the software is no longer considered the prototype, but the production model, and its life cycle begins.

f. Operation

After implementation, DoD STD-2167A currently considers the system complete. This perception is one of the major drawbacks to the standard when considering that the maintenance or Operation activity of a software product's life is taking up to 80 percent of analysts' energies.

DoD STD-2167A must address the Operation activity. Since maintenance encompasses both error correction and software upgrades, detected errors are to be corrected. On the other hand, enhancements and changes in user requirements must be cycled back through the Analysis activity for verification that the system can be modified with the constraints previously built-in.

Changes to the software system must be cycled through the complete series of steps to ensure that documentation is up to date. The IRDS will be enhanced as business policies change, and the software system will reflect those changes as long as the changes are implemented by going through the complete cycle. This is basically the concept of reverse engineering as examined in Chapter two.

The software will continue to serve a useful life as long as changes can be made that allow it to function under the requirements stated in the System Requirements Document. As soon as the software can no longer be modified under the imposed constraints or those constraints change, the cycle must begin again at the Study activity. Due to the continued expansion of the IRDS, subsequent development activities will be able to utilize an increasing library of software modules and data definitions. With the IRDS library available, the Study activity should become similar to the development of a new automobile - use old modules and data

definitions where applicable, create new modules where needed, and assemble them into a new product.

2. DoD STD-2167A using CASE without IRDS

NAVSUP's experience developing and using an IRDS suggests that initial use of CASE tools for software development may be more successful without an IRDS. The development of the NAVSUP IRDS has yet to contribute any results to the overall operation in terms of operational efficiency, cost savings, or increased software development productivity. Therefore, over the three year period that strategic planning and business policies have been documented, IRDS development has been done with a constantly decreasing budget [Ref. 20:p. 50].

In an organization without an IRDS, the activities followed in developing software systems are no different than those described for organizations with an IRDS. The difference is in the initial approach to selection of software projects and the use of the CASE data encyclopedia as the backbone of the IRDS.

a. Study Activity

The first activity in development of a software system is the Study activity. Without the benefit of an IRDS, the organization must begin by utilizing a CASE tool to initiate an Information Engineering approach to development of the corporate IRDS.

Using the Data Encyclopedia as the first iteration of an IRDS, the organization should document the corporate strategy and develop a skeleton of its various business policies. The analyst and management should select a business activity that will be positively impacted by the development of a new software system [Ref. 17]. The business activity selected will be further studied and a single software application chosen as a research project for utilizing this development methodology. The analyst will create an SDP with initial cost-benefit documents for the project. The SDP should be reviewed at a System Requirements Review between developers, users and management.

The success of the first project is at least as important as the design benefits achieved by utilizing the methodology. Its success will enable the halo effect to encircle future projects.

The developers should choose a project that has a high probability of success and the project should be able to produce a useful software product in a short period of time - six to nine months. Senior management should support the project from its inception. The end users of the project must have a stake in its successful outcome and must work closely with the developers to avoid failure.

As an example of how not to approach a project, NAVSEA's use of USER:Expert System™ CASE toolkit for developing a corporate Data Dictionary was a failure in 1987

mainly because the project produced no useful end user information and had little senior management support. As a result, NAVSEA has yet to attempt another venture into the use of CASE tools.

b. Analysis Activity

The Analysis activity will involve a detailed investigation of what the users need to fulfill the business requirements established in the Study activity. Using a CASE diagramming tool, the analyst and users will develop a complete logical model detailing the environmental model and the behavioral model. The development of the behavioral model will record data definitions, processes, and relationships in the Data Encyclopedia which will become the IRDS at a later date.

Using the logical model and the SDP, the analyst should develop new physical models for the user to review. The users and management will then choose between the options presented. Software Requirements Specification and Interface Requirements Specification for the chosen model will be accepted as the Allocated Baseline at a System Specifications Review.

c. Further Activities

The activities that follow the Analysis activity in the development cycle will proceed exactly as described in sections (c) through (f) of **DoD STD-2167A using CASE with IRDS**. However, each time the IRDS is mentioned the term Data

Encyclopedia could be substituted. Subsequent development projects will add to the Data Encyclopedia or utilize previously stored Data Encyclopedia information. The continued utilization of a single Data Encyclopedia system will allow the organization to develop an integrated IRDS.

D. SUMMARY

DoD STD-2167A can be tailored to meet the methodology of the software developers. By using CASE tools with DoD STD-2167A, the software development cycle can be impacted. CASE tools that produce DIDs readily fit into the standard.

Use of CASE tools with DoD STD-2167A enables organizations to utilize the elements of an IRDS and verify the contents of the IRDS. If an organization does not have an IRDS in place, using CASE tools in the development of their software systems can help them to develop an IRDS.

Flaws in DoD STD-2167A that hinder user-developer interface are negated by the use of CASE tools throughout the development cycle because users and designers are in constant contact analyzing the system and reviewing the way it is designed. Problems with the bottom-up design and implementation are addressed by CASE tools that automatically create structure charts from design diagrams. This allows the designer to choose a top-down coding and implementation scheme. Problems with sequentially following a pattern are reduced by ensuring that all development efforts conform to

data contained in the Data Encyclopedia and are verified against Software and Interface Requirements Specifications.

The life of software products is addressed by introducing CASE tools to DoD STD-2167A. Data maintained in the Data Encyclopedia becomes the nucleus to control software during the Operation activity.

V. CONCLUSIONS

A. DoD STD-2167A AND CASE

This thesis has covered the background of DoD STD-2167A, the Systems Development Life Cycle, and Computer Aided Software Engineering tools. A tailored version of DoD STD-2167A has been proposed in Chapter IV that will allow the successful integration of CASE tools into the methodology.

DoD STD-2167A and SDLC rely on following a structured procedure to develop software systems. The use of waterfall development techniques depend on following structured analysis, structured design, and a step-by-step approach to bottom-up development, testing, and implementation.

The proposed tailoring of DoD STD-2167A favors the top-down approach to development and implementation endorsed by Ed Yourdon [Ref. 19:pp 42-64]. Top-down development will allow all interface modules to be developed and tested first. It also will provide for implementation to be done on an incremental basis, because the skeleton system can be installed before all operational modules are completed by using dummy modules in the place of operational modules. Integrated testing is accomplished using the skeleton system interfaces to outside systems. As the lower level modules are completed they will replace the dummies without having to go through another integrated testing procedure.

DoD STD-2167A and SDLC are both considered successful approaches to software development, yet there remains a large backlog in unfinished software development projects. Success of the waterfall methods is largely measured by empirical studies of user satisfaction and a backlog that is increasing at a slower rate than the increase in demand for new software.

DoD STD-2167A is encumbered by the requirement to completely document the traceability of software modules to the software development requirements. The documentation involved includes the completion of Data Item Descriptions that are examined and audited at formal developer/management reviews. CASE tools that automatically generate the documentation in the required format should produce development time shrinkages.

CASE tools are touted to increase productivity through consistent application of a single methodology throughout the development cycle of software products. They will achieve greater improvements in productivity and quality once CASE tools are integrated and designers rely on a single Data Encyclopedia for all development efforts. The Data Encyclopedia will either become the basis for organization wide Information Resource Directory Systems (IRDS) or will be able to draw from the IRDS to ensure consistent application development that reflects the corporate strategy and business policies.

The proposed tailoring of DoD STD-2167A in Chapter IV to incorporate CASE tools is consistent with the intent of the standard to modify it to make use of modern productivity tools. An operational activity has been added to the proposed standard to allow modification of completed applications, but requiring that the modifications be consistent with the business policies and system requirements that are stored in the IRDS.

B. AUTOMATED TAILORING OF DoD STD-2167A

The Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management has entered into a licensing agreement with Logicon, Inc. to develop an automated means to tailor DoD STD-2167A. Tailor™ is available free of charge to authorized DoD users.

Tailor's™ main strength is that it allows users to tailor DoD STD-2167A by proceeding through a series of menu driven questions designed to consistently apply the nearly 250 requirements. Its secondary strength is that it reduces the time required to tailor DoD STD-2167A from two weeks when done manually to two hours when done with the software. Tailor's™ major weakness is that it specifically addresses only the functional paragraphs of the standard and does not guide the user on how or where to apply productivity improvement tools such as prototype generators and CASE tools.

C. AREAS FOR FURTHER RESEARCH

1. Test of Proposed DoD STD-2167A

The proposed tailoring of DoD STD-2167A to include CASE tools could significantly impact organizations that develop software under the constraints of the standard. When CASE tools become true I-CASE toolkits, the Data Encyclopedia will become one of the major information depositories of the organization.

Fleet Numerical Oceanagraphics Center, Monterey California (FNOC) has been investigating a specific tailoring of DoD STD-2167A to their organizational needs. Further research on how the proposed tailoring of DoD STD-2167A to include CASE tools would be applicable to FNOC is encouraged.

2. British Aerospace Australia

CASE tools are currently being utilized on a real-time system development for the Royal Australian Navy. The project is contracted to British Aerospace Australia, and the contract stipulated that the development follow DoD STD-2167. John Viskic of British Aerospace will be conducting post-project analysis of productivity improvements on completion in August 1989. The results of the analysis could be made available to assist in further research on how CASE should be utilized under the constraints of DoD STD-2167A.

3. Productivity Measures

It has been contended that CASE tools will increase the productivity in software development and quality of

software products. A significant problem is that measures of productivity and quality are used by very few corporations [Ref. 7]. A significant number of metrics have been proposed to capture the productivity and quality information, but the historical data base of previous development efforts on which to make comparisons does not exist.

The initial means by which measurement of CASE productivity improvements will have to be made is against estimates of the project development effort, ie., "Was the project completed on time? Were there fewer errors than estimated?". When a project is started using CASE tools, it will be imperative that the developers maintain project measures in the Data Encyclopedia as part of a project data base. The only means by which comparisons on productivity and quality improvements can be made is to ensure that during development all projects maintain a project data base.

Further research is required in the areas of productivity improvement and quality improvement by developments using CASE. Measurement standards need to be applied to test cases for development efforts using and not using CASE.

4. Expectations for CASE

The initial impetus of this research effort was to explore what end-users should expect from applications developed using CASE tools. The problems encountered by the author included:

1. Most organizations contacted, if using CASE, were using it on an experimental basis.

2. CASE tools are very expensive to purchase and considered luxury items.

3. CASE technology is changing rapidly and organizations don't want to be purchasing a product that will be antiquated before they have recovered their investment.

4. CASE tools are not integrated to a large enough extent to allow the transportability of software design information through the entire life cycle.

As CASE tools become more prevalent in software development, further research is required to answer how CASE will effect the products delivered to the end-user. The end-user should be able to understand whether using CASE will produce a higher quality software product, whether the product will be delivered in a timely manner, and most important whether the product will solve his business needs and contribute to the corporate strategy.

LIST OF REFERENCES

1. Martin, James, "CASE & I-CASE," High-Productivity Software, Inc., Marblehead, Massachusetts, 1988.

2. Mosley, Daniel, "CASE Tools: The Future of Applications Development," ShowCASE Conference III, The Center for the Study of Date Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.

3. Martin, James, "The Future of CASE Technology," ShowCASE Conference III, The Center for the Study of Data Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.

4. Boehm, Barry and Standish, Thomas, "Software Technology in the 1990's: Using an Evolutionary Paradigm," IEEE Computer, Vol. 16, No. 11, November 1983.

5. Sprague, Ralph and McNurlin, Barbara, Information Systems Management in Practice, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

6. Necco, Charles, Gordon, Carl, and Tsai, Nancy, "Systems Analysis and Design: Current Practices," MIS QUARTERLY, Vol. 11, No. 4, December 1987.

7. Jones, Capers, "The Cost and Value of CASE," ShowCASE Conference III, The Center for the Study of Data Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.

8. Pressman, Roger, Software Engineering, A Practioner's Approach, McGraw-Hill Book Company, New York, New York, 1987.

9. Wallace, Steve, "Methodology: CASE's Critical Cornerstone," Business Software Review, Vol. 7, No. 4, April 1988.

10. Merlyn, Vaughan, "The Impact of CASE on Quality," ShowCASE Conference III, The Center for the Study of Data Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.

11. Gane, Chris, "The Impact of Expert Systems Technology on CASE Products," ShowCASE Conference III, The Center for the Study of Data Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.

12. Stanley, Sharon, Information Engineering in the Department of Defense: Two Case Studies, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1988.

13. Fisher, Alan, CASE, using Software Development Tools, John Wiley & Sons, Inc., New York, New York, 1988.

14. Hanner, Mark, "CASE TOOLS Productivity for the Masses," DEC Professional, Vol. 7, No. 12, December 1988.

15. Brooks, Frederick, The Mythical Man Month, Addison-Wesley Publishing Company, Reading, Massachusetts, 1975.

16. Bachman, Charlie, "A CASE for Reverse Engineering," DATAMATION, Vol. 34, No. 13, July 1, 1988.

17. Shultz, Scott, "CASE as a Strategic Weapon," ShowCASE Conference III, The Center for the Study of Data Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.

18. Department of Defense Military Standard DoD STD-2167A, DEFENSE SYSTEM SOFTWARE DEVELOPMENT, 4 June 1985.

19. Yourdon, Edward, Managing the System Life Cycle, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

20. Barber, Mark and Richey, Paul, NAVSUP DA:Planning and Implementation, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6154	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Computer Technology Curricular Office, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
4.	Department of Administative Sciences Naval Postgraduate School Professor Barry A. Frew, 54Fw Monterey, California 93943-5000	1
5.	Department of Administative Sciences Naval Postgraduate School Professor James E. Suchan, 54Sa Monterey, California 93943-5000	1
6.	Fleet Numerical Oceanagraphic Center, Code 008 Monterey, California 93940	1
7.	Naval Data Automation Command Washington Navy Yard Attn: LT Gary Batt Washington, D.C. 20374-1662	1

20 OCT 92
4 OCT 92
12 OCT 92
38497
38497
3850n
12 JAN 94
JAN - 4 1995
3
GAYLORD #3
Keep this card in the book pocket
Book is due on the latest date stamped

Tl esis
B2425635 Batt
c.1 CASE technology and
the Systems Development Life Cycle.

thesB2425635
CASE technology and the Systems Developm
3 2768 000 81209 3
DUDLEY KNOX LIBRARY